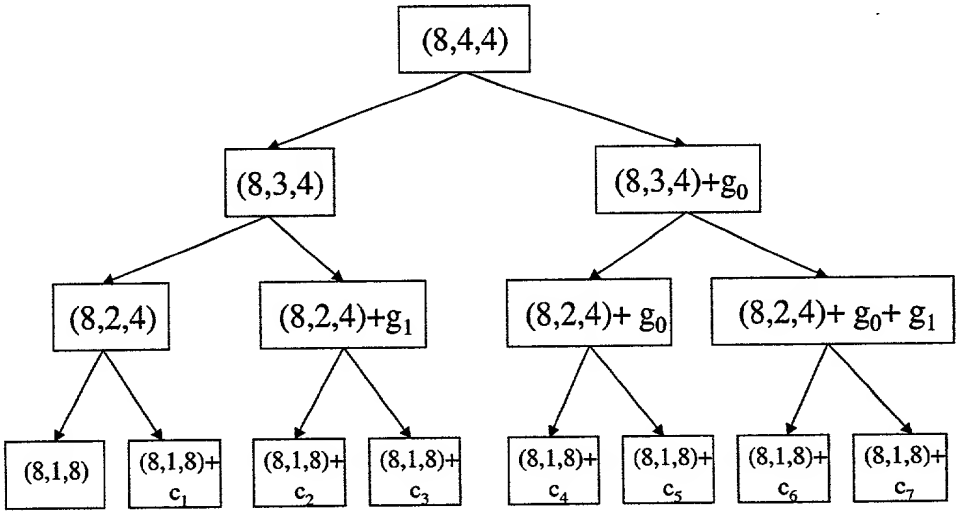
APPENDIX A

Multidimensional Reed-Muller Codes

This section derives the multidimensional $(8,4,4)$ Reed-Muller codes using the set partitioning techniques in cite: pietrobon1 , cite: pietrobon2 . This technique does set partitioning on Reed-Muller code spaces instead of lattices to build code spaces. Two-level set partitioning can be applied to an $(8,4,d_{\min}=4)$ Reed-Muller (i.e. $\mathcal{RM}(1,3)$) code as shown in figure (ref: fig:codepart) to yield subcode partition chains cite: forney3 of $(8,4,4) \supset (8,3,4) \supset (8,2,4) \supset (8,1,8) = \mathcal{RM}(0,3)$. The Hamming weight of a codeword is denoted as w_H. At level 1 of the 2-way subgroup code partition, g_0 is the coset leader that is added to (8,3,4) Reed-Muller sub-code to form an (8,4,4) Reed-Muller codeword or $C_{(8,4,4)} = C_{(8,3,4)} \oplus a_i g_0$ where $a_i \in \{0,1\}$ and $g_0 \notin C_{(8,3,4)}$ and $g_0 \in C_{(8,4,4)}$. Likewise the level 2, 2-way subgroup code partition has the form $C_{(8,4,4)} = C_{(8,2,4)} \oplus \sum_{i=0}^{1} a_i g_i$ where $a_i \in \{0,1\}$ and $g_i \notin C_{(8,2,4)}$ and $g_i \in C_{(8,3,4)}$ or $g_i \in C_{(8,4,4)}$. The level 3, 2-way subgroup code partition has the form $C_{(8,4,4)} = C_{(8,1,8)} \oplus \sum_{i=0}^{2} a_i g_i$ where $a_i \in \{0,1\}$ and $g_i \notin C_{(8,1,8)}$ and $g_i \in C_{(8,2,4)}$ or $g_i \in C_{(8,3,4)}$ or $g_i \in C_{(8,4,4)}$. Each vector $c_j = \sum_{i=0}^{2} a_i g_i$ where $j = (a_2, a_1, a_0)_8$.



Subgroup code partitions for (8,4,4) Reed-Muller code

Notice that 3 levels of set partitioning yields a repetition code $(8,1,8)$ with $d_{\min} = 8$. It will be shown that a simple trellis coded, 2-dimensional $(8,4,4)$ Reed-Muller code can yield a new code with $d_{\min} \geq 2 \times 8 = 16$ or a potential coding gain of 6 dB. over a $(8,4,4)$ Reed-Muller code.

Table (ref: tab:map) shows the assignment of codewords to branch labels. The branch label assignment tries to preserve the conventional labeling schemes for typical modulations such as QAM and M-PSK. For example, for 16-QAM a pair of constellations points $(0,2)$ and $(0,4)$ represent a sequence of pairs of points that are separated further in Euclidean distance because the numerical difference in the points is larger (i.e. $(0,4)$ has larger Euclidean distance than $(0,2)$). Likewise $(0,8)$ has large Euclidean distance than $(0,4)$ and $(0,2)$. However the Euclidean distance of 16 constellation points of an $(8,4,4)$ Reed-Muller code space have three distinct Euclidean distances which are associated with the codewords having w_H as shown in table (ref: tab:map). Therefore the branch labeling odes not have exactly the same meaning as for conventional modulation constellations. For example, the branch label difference between $(1,3)$ or $(1,4)$ does not reflect that

the Euclidean distances between these pairs of points are the same. However there is some significance for branch labeling when looking at pairs of constellation points that are numerically separated by 8 in branch labeling. Pairs (0,8) and (1,9) have $d_{\min} = 8$ in Hamming distance per dimension. The pairs of codewords associated with the labels are also Boolean complements of each other. This property is important when performing set partitioning to yield constellation points with Euclidean distances larger than the codewords in an underlying (8,4,4) Reed-Muller code.

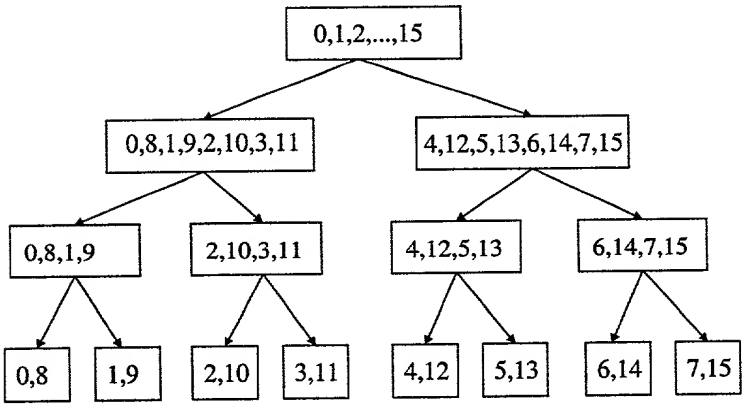
[B]

Binary Codeword	Constellation Point Label	w_H
(0 0 0 0 0 0 0 0)	0	0
(0 1 0 1 0 1 0 1)	1	4
(0 0 1 1 0 0 1 1)	2	4
(0 1 1 0 0 1 1 0)	3	4
(0 0 0 0 1 1 1 1)	4	4
(0 1 0 1 1 0 1 0)	5	4
(0 0 1 1 1 1 0 0)	6	4
(0 1 1 0 1 0 0 1)	7	4
(1 1 1 1 1 1 1 1)	8	8
(1 0 1 0 1 0 1 0)	9	4
(1 1 0 0 1 1 0 0)	10	4
(1 0 0 1 1 0 0 1)	11	4
(1 1 1 1 0 0 0 0)	12	4
(1 0 1 0 0 1 0 1)	13	4
(1 1 0 0 0 0 1 1)	14	4
(1 0 0 1 0 1 1 0)	15	4

caption [E]

Using the branch labels in figure (ref: fig:1Dsetpart), the set partitioning for an (8,4,4) Reed-Muller set of codeword "constellation" points is shown. A constellation point in Euclidean space is formed by transforming the binary codewords with $b_{i,k} = (-1)^{c_{i,k}}$ where $c_{i,k}$ is a bit in a binary codeword c_i. Now the Euclidean distance between two codewords c_i, c_j is defined as $d^2(c_i, c_j) = \sum_k (b_{i,k} - b_{j,k})^2$. Notice that the minimum *Euclidean* distance *between* codeword constellation points $\delta^2_{\min}$ at each partition level j is denoted as $\delta^2_{j,\min}$ therefore $\delta^2_{0,\min} = 16 \leq \delta^2_{1,\min} = 16 \leq \delta^2_{2,\min} = 16 < \delta^2_{3,\min} = 32$ where partition level 0 includes all codeword labels $(0, 1, \ldots, 15)$

Set partitioning



Set partitioning for 1D (8,4,4) Reed-Muller code

Table (ref: tab:trellis1Drm) shows the set partitions for a 2-D (8,4,4) Reed-Muller code. To represent a $2 \times (8,4,4)$ Reed-Muller code, a 2×4 binary matrix is formed

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_1^3 & y_1^2 & y_1^1 & y_1^0 \\ y_2^3 & y_2^2 & y_2^1 & y_2^0 \end{bmatrix} \quad \#$$

A total of $2^8 = 256$ points (i.e. codeword constellation points) represent the unpartitioned signal set. Each 2-tuple $[y_1, y_2]^T$ has two entries that represent a branch (i.e. Reed-Muller codeword) label pair in table (ref: tab:map). Each branch label can also be represented in binary form in equation (ref: eq:2tuple) where $y_i^j \in \{0,1\}$ and $y_i = 8y_j^3 + 4y_j^2 + 2y_j^1 + y_j^0$. The same mechanism for set partitioning cite: pietrobon2 is used for a 2×8-PSK example with codes and subcodes of a $(2,2,1)$ linear block code. Therefore C_0 is a $(2,2,1)$ block code and C_1, C_2 are subcodes $(2,1,2)$ and $(2,0,\infty)$ respectively of C_0 and $C_0 \supset C_1 \supset C_2$.

This construction has similarities to the treatment of 16 QAM in cite: pietrobon1 since there are 16 (8,4,4) Reed-Muller codewords ("constellation" points). Using the same notation, the minimum squared subset distance (MSSD) of a partition is defined as $\Delta_p^2 \geq \min(\delta_{l-1}^2 d_{m_{l-1}}, \delta_{l-2}^2 d_{m_{l-2}}, \delta_{l-3}^2 d_{m_{l-3}}, \ldots, \delta_0^2 d_{m_0})$ where l is the set partitioning level of a Reed-Muller code as shown in figure ref: fig:codepart , m_l is the index of binary code C_{m_l} used to provide the set partitioning at level l, d_{m_l} is the Hamming distance of the codewords being used to provide set partitioning and δ_l^2 is the Euclidean distance between constellation points at level l of the 1-D constellation points for the Reed-Muller code. Notice that l corresponds to a bit y_j^l in the binary representation of a branch label in equation (ref: eq:2tuple). For example, to compute the MSSD with $d_{C_0} = 1, d_{C_1} = 2, d_{C_2} = \infty$, then $\Delta_p^2 \geq \min(32d_3, 16d_2, 16d_1, 16d_0)$ and for partition at level 0, $\Delta_0^2 \geq \min(32d_{C_0}, 16d_{C_0}, 16d_{C_0}, 16d_{C_0}) = \min(32, 16, 16, 16) = 16$. The remaining entries at all other partition levels are shown in table (ref: tab:trellis1Drm). Notice that it is necessary to partition down to lowest levels $(\text{i.e. } p = 5,6,7)$ to get separation between codewords in Euclidean space to get the benefits of trellis coding. (The 1-D codewords have a separation of $\Delta^2 = 32$.) It will be shown that the number of encoder memory stages must be at least 3 (i.e. 8 states of memory) to get any benefits of this encoding.

Level (p)	Partition Ω^p	MSSD (Δ_p^2)	Generator $(t_p)^T$
0	$\Omega(C_0, C_0, C_0, C_0)$	$\min(32, 16, 16, 16) = 16$	$\begin{bmatrix} 0 & 1 \end{bmatrix}$
1	$\Omega(C_0, C_0, C_0, C_1)$	$\min(32, 16, 16, 32) = 16$	$\begin{bmatrix} 1 & 1 \end{bmatrix}$
2	$\Omega(C_0, C_0, C_0, C_2)$	$\min(32, 16, 16, \infty) = 16$	$\begin{bmatrix} 0 & 2 \end{bmatrix}$
3	$\Omega(C_0, C_0, C_1, C_2)$	$\min(32, 16, 32, \infty) = 16$	$\begin{bmatrix} 2 & 2 \end{bmatrix}$
4	$\Omega(C_0, C_0, C_2, C_2)$	$\min(32, 16, \infty, \infty) = 16$	$\begin{bmatrix} 0 & 4 \end{bmatrix}$
5	$\Omega(C_0, C_1, C_2, C_2)$	$\min(32, 32, \infty, \infty) = 32$	$\begin{bmatrix} 4 & 4 \end{bmatrix}$
6	$\Omega(C_0, C_2, C_2, C_2)$	$\min(32, \infty, \infty, \infty) = 32$	$\begin{bmatrix} 0 & 8 \end{bmatrix}$
7	$\Omega(C_1, C_2, C_2, C_2)$	$\min(64, \infty, \infty, \infty) = 64$	$\begin{bmatrix} 8 & 8 \end{bmatrix}$
8	$\Omega(C_2, C_2, C_2, C_2)$	$\min(\infty, \infty, \infty, \infty) = \infty$	-

[B] caption [E]

Table (ref: tab:trellis1Drm_2) shows a different set partitioning of Ω^0. The number of encoder memory stages must be at least 3 (i.e. 8 states of memory) to get any benefits of this encoding. For this example, there are more partitions with MSSD > 16 than the previous case. A trellis code developed with this partitioning scheme should have better distance properties for the same number of trellis states.

The cosets for partitions Ω^p can be formed as shown in cite: pietrobon2 . For example, $\Omega^1(z^0) = \Omega^1 + z^0 t^0 \bmod(16)$ and $\Omega^2(2z^1 + z^0) = \Omega^2 + 2z^1 t^1 + z^0 t^0$ where $z^i \in \{0, 1\}$. Using entries in table (ref: tab:trellis1Drm),

$$\Omega^2(2z^1 + z^0) = \Omega^2 + 2z^1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + z^0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \bmod(16)$$

and in general a new codeword $y(z) = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \Omega^8(z)$ in 2-space is formed as follows

$$\begin{aligned} y(z) = {} & z^7 \begin{bmatrix} 8 \\ 8 \end{bmatrix} + z^6 \begin{bmatrix} 0 \\ 8 \end{bmatrix} + z^5 \begin{bmatrix} 4 \\ 4 \end{bmatrix} + z^4 \begin{bmatrix} 0 \\ 4 \end{bmatrix} \\ & + z^3 \begin{bmatrix} 2 \\ 2 \end{bmatrix} + z^2 \begin{bmatrix} 0 \\ 2 \end{bmatrix} + z^1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + z^0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \bmod(16) \end{aligned} \quad \#$$

All tuples are formed as shown in equation (ref: eq:code2D). Note that partition $\Omega^8(0)$ is the all "0" codeword or "constellation" point $[0, 0]^T$. The encoding is formed using a trellis encoder for 8 bits z^i. Note also that the 8 bits in $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ are *not* arbitrarily assigned to the 8 bits in z. A signal set mapper is used for the assignment which is discussed in detail in section ref: sect:design .

So for $z = [z^7, z^6, ..., z^0] = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$ then

$y(z) = \begin{bmatrix} 0 \\ 8 \end{bmatrix} + \begin{bmatrix} 0 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 13 \end{bmatrix}$. The encoding process selects a Reed-Muller 2-tuple of codewords which correspond to $y(z) = \begin{bmatrix} (0\ 1\ 0\ 1\ 0\ 1\ 0\ 1) \\ (1\ 0\ 1\ 0\ 0\ 1\ 0\ 1) \end{bmatrix}$

The encoding process is a two step process. First the information bits are mapped from $IL - 1$ bits to IL bits which represent z^i bits that select a particular coset of Ω^8. Using the same notation as in cite: pietrobon2 , I is the number of information bits and L is the dimensionality of the multidimensional Reed-Muller code. The next step is to map the intermediate information bits to Reed-Muller codewords. The code rate for the new encoding process actually maps $IL - 1$ information bits to $I'L$ bits where I' is the number of encoded bits in a Reed-Muller codeword and L is the dimensionality of the multidimensional Reed-Muller code. In general this yields a code of rate $R = \frac{IL-1}{IL} \times \frac{IL}{I'L} =$. For the case $I = 4, I' = 8$ and $L = 2$ then $R = \frac{7}{16}$. For $L = 3$, the code rate is $R = \frac{11}{24}$. Notice that the overall code rate is less than the code rates for 2 or 3 successive Reed-Muller codewords respectively (i.e. $R = 8/16$ and $R = 12/24$). Trellis encoding is used to (1) introduce redundancy beyond that of the information bits of original codeword and (2) provide the memory necessary to construct the trellises to give additional coding gain. In order to construct Reed-Muller codewords for each dimension it is necessary to have 1 less information bit at the input to the multidimensional encoder since the trellis encoder is a rate $\frac{k}{k+1}$ encoder where $k \leq IL - 1$. If it is desired to preserve an overall code rate $R = 1/2$ then a trellis encoder of rate $\frac{IL}{IL+1}$ can be used with puncturing applied to the output of the encoder to give a rate of "1". This development will nonetheless describe the workings of the trellis encoding using the original constructions in cite: pietrobon2 and then describe the performance degradations due to puncturing to preserve the code rates of multiple "1-D" Reed-Muller codes.

[B]

Level (p)	Partition Ω^p	MSSD (Δ_p^2)	Generator $(t_p)^T$
0	$\Omega(C_0, C_0, C_0, C_0)$	$\min(32, 16, 16, 16) = 16$	$\begin{bmatrix} 0 & 1 \end{bmatrix}$
1	$\Omega(C_0, C_0, C_0, C_1)$	$\min(32, 16, 16, 32) = 16$	$\begin{bmatrix} 0 & 2 \end{bmatrix}$
2	$\Omega(C_0, C_0, C_1, C_1)$	$\min(32, 16, 32, 32) = 16$	$\begin{bmatrix} 0 & 4 \end{bmatrix}$
3	$\Omega(C_0, C_1, C_1, C_1)$	$\min(32, 32, 32, 32) = 32$	$\begin{bmatrix} 1 & 1 \end{bmatrix}$
4	$\Omega(C_0, C_1, C_1, C_2)$	$\min(32, 32, 32, \infty) = 32$	$\begin{bmatrix} 2 & 2 \end{bmatrix}$
5	$\Omega(C_0, C_1, C_2, C_2)$	$\min(32, 32, \infty, \infty) = 32$	$\begin{bmatrix} 4 & 4 \end{bmatrix}$
6	$\Omega(C_0, C_2, C_2, C_2)$	$\min(32, \infty, \infty, \infty) = 32$	$\begin{bmatrix} 0 & 8 \end{bmatrix}$
7	$\Omega(C_1, C_2, C_2, C_2)$	$\min(64, \infty, \infty, \infty) = 64$	$\begin{bmatrix} 8 & 8 \end{bmatrix}$
8	$\Omega(C_2, C_2, C_2, C_2)$	$\min(\infty, \infty, \infty, \infty) = \infty$	-

caption [E]

Each partition is enumerated using a matrix representation. Thus Ω^0 which is the partition with all pairs of branch labels is a (16×16) matrix of 2-tuples.

$$\Omega^0 = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \begin{bmatrix} 0 \\ 2 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 15 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 2 \end{bmatrix} & \cdots & \begin{bmatrix} 1 \\ 15 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 0 \end{bmatrix} & \begin{bmatrix} 2 \\ 1 \end{bmatrix} & \begin{bmatrix} 2 \\ 2 \end{bmatrix} & \cdots & \begin{bmatrix} 2 \\ 15 \end{bmatrix} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \begin{bmatrix} 15 \\ 0 \end{bmatrix} & \begin{bmatrix} 15 \\ 1 \end{bmatrix} & \begin{bmatrix} 15 \\ 2 \end{bmatrix} & \cdots & \begin{bmatrix} 15 \\ 15 \end{bmatrix} \end{bmatrix}$$

The 2-way partition Ω^1 has 128 entries and represents all 2-tuples with even or odd pairs of branch labels. There are 64 even pairs and 64 odd pairs of tuples. The other partiton at level 1 is formed by adding t^0 to Ω^1 or $\Omega^1(1) = \Omega^1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

$$\Omega^1 = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 2 \end{bmatrix} & \begin{bmatrix} 0 \\ 4 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 14 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 0 \end{bmatrix} & \begin{bmatrix} 2 \\ 2 \end{bmatrix} & \begin{bmatrix} 2 \\ 4 \end{bmatrix} & \cdots & \begin{bmatrix} 2 \\ 14 \end{bmatrix} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \begin{bmatrix} 14 \\ 0 \end{bmatrix} & \begin{bmatrix} 14 \\ 2 \end{bmatrix} & \begin{bmatrix} 14 \\ 4 \end{bmatrix} & \cdots & \begin{bmatrix} 14 \\ 14 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 3 \end{bmatrix} & \begin{bmatrix} 1 \\ 5 \end{bmatrix} & \cdots & \begin{bmatrix} 1 \\ 15 \end{bmatrix} \\ \begin{bmatrix} 3 \\ 1 \end{bmatrix} & \begin{bmatrix} 3 \\ 3 \end{bmatrix} & \begin{bmatrix} 3 \\ 5 \end{bmatrix} & \cdots & \begin{bmatrix} 3 \\ 15 \end{bmatrix} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \begin{bmatrix} 15 \\ 1 \end{bmatrix} & \begin{bmatrix} 15 \\ 3 \end{bmatrix} & \begin{bmatrix} 15 \\ 5 \end{bmatrix} & \cdots & \begin{bmatrix} 15 \\ 15 \end{bmatrix} \end{bmatrix}$$

The 2-way partition Ω^2 has 64 entries and represents all 2-tuples with only even pairs of branch labels. The other partitions at level 2 is formed as $\Omega^2(z) = \Omega^2 + z^1\begin{bmatrix} 1 \\ 1 \end{bmatrix} + z^0\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Therefore

$\Omega^2(2) = \Omega^2 + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Thus $\Omega^2(2)$ is a partition of all odd pairs of tuples in Ω^1

$$\Omega^2 = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 2 \end{bmatrix} & \begin{bmatrix} 0 \\ 4 \end{bmatrix} & \cdots & \begin{bmatrix} 0 \\ 14 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 0 \end{bmatrix} & \begin{bmatrix} 2 \\ 2 \end{bmatrix} & \begin{bmatrix} 2 \\ 4 \end{bmatrix} & \cdots & \begin{bmatrix} 2 \\ 14 \end{bmatrix} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \begin{bmatrix} 14 \\ 0 \end{bmatrix} & \begin{bmatrix} 14 \\ 2 \end{bmatrix} & \begin{bmatrix} 14 \\ 4 \end{bmatrix} & \cdots & \begin{bmatrix} 14 \\ 14 \end{bmatrix} \end{bmatrix}$$

Carrying the partitioning to the next level, the 2-way partition Ω^3 has 32 entries and further partitions branch labels which are even. The other partitions at level 3 is formed as $\Omega^3(4z^2 + 2z^1 + z^0) = \Omega^3 + z^2\begin{bmatrix} 0 \\ 2 \end{bmatrix} + z^1\begin{bmatrix} 1 \\ 1 \end{bmatrix} + z^0\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Therefore $\Omega^3(4) = \Omega^3 + \begin{bmatrix} 0 \\ 2 \end{bmatrix}$. Thus $\Omega^3(4)$ is a partition of all even pairs of tuples in Ω^2. The new matrix representation is a "checkerboard" pattern which is a (8×4) matrix

$$\Omega^3 = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 4 \end{bmatrix} & \begin{bmatrix} 0 \\ 8 \end{bmatrix} & \begin{bmatrix} 0 \\ 12 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 2 \end{bmatrix} & \begin{bmatrix} 2 \\ 6 \end{bmatrix} & \begin{bmatrix} 2 \\ 10 \end{bmatrix} & \begin{bmatrix} 2 \\ 14 \end{bmatrix} \\ \begin{bmatrix} 4 \\ 0 \end{bmatrix} & \begin{bmatrix} 4 \\ 4 \end{bmatrix} & \begin{bmatrix} 4 \\ 8 \end{bmatrix} & \begin{bmatrix} 4 \\ 12 \end{bmatrix} \\ \begin{bmatrix} 6 \\ 2 \end{bmatrix} & \begin{bmatrix} 6 \\ 6 \end{bmatrix} & \begin{bmatrix} 6 \\ 10 \end{bmatrix} & \begin{bmatrix} 6 \\ 14 \end{bmatrix} \\ \vdots & \vdots & \vdots & \vdots \\ \begin{bmatrix} 14 \\ 2 \end{bmatrix} & \begin{bmatrix} 14 \\ 6 \end{bmatrix} & \begin{bmatrix} 14 \\ 10 \end{bmatrix} & \begin{bmatrix} 14 \\ 14 \end{bmatrix} \end{bmatrix}$$

The partition at level 4 is Ω^4 and has 16 elements.

$$\Omega^4 = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 4 \end{bmatrix} & \begin{bmatrix} 0 \\ 8 \end{bmatrix} & \begin{bmatrix} 0 \\ 12 \end{bmatrix} \\ \begin{bmatrix} 4 \\ 0 \end{bmatrix} & \begin{bmatrix} 4 \\ 4 \end{bmatrix} & \begin{bmatrix} 4 \\ 8 \end{bmatrix} & \begin{bmatrix} 4 \\ 12 \end{bmatrix} \\ \begin{bmatrix} 8 \\ 0 \end{bmatrix} & \begin{bmatrix} 8 \\ 4 \end{bmatrix} & \begin{bmatrix} 8 \\ 8 \end{bmatrix} & \begin{bmatrix} 8 \\ 12 \end{bmatrix} \\ \begin{bmatrix} 12 \\ 0 \end{bmatrix} & \begin{bmatrix} 12 \\ 4 \end{bmatrix} & \begin{bmatrix} 12 \\ 8 \end{bmatrix} & \begin{bmatrix} 12 \\ 12 \end{bmatrix} \end{bmatrix}$$

$\Omega^4(8z^3 + 4z^2 + 2z^1 + z^0) = \Omega^4 + z^3\begin{bmatrix} 2 \\ 2 \end{bmatrix} + z^2\begin{bmatrix} 0 \\ 2 \end{bmatrix} + z^1\begin{bmatrix} 1 \\ 1 \end{bmatrix} + z^0\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Therefore $\Omega^4(8) = \Omega^4 + z^3\begin{bmatrix} 2 \\ 2 \end{bmatrix}$. Notice that $\Omega^3 = \Omega^4 \cup \Omega^4(8)$. The remaining partitions are formed as follows, Ω^5 is a (4×2) matrix of elements

$$\Omega^5 = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 8 \end{bmatrix} \\ \begin{bmatrix} 4 \\ 4 \end{bmatrix} & \begin{bmatrix} 4 \\ 12 \end{bmatrix} \\ \begin{bmatrix} 8 \\ 0 \end{bmatrix} & \begin{bmatrix} 8 \\ 8 \end{bmatrix} \\ \begin{bmatrix} 12 \\ 4 \end{bmatrix} & \begin{bmatrix} 12 \\ 12 \end{bmatrix} \end{bmatrix}$$

and

$\Omega^5(16z^4 + 8z^3 + 4z^2 + 2z^1 + z^0) = \Omega^4 + z^4\begin{bmatrix} 0 \\ 4 \end{bmatrix} + z^3\begin{bmatrix} 2 \\ 2 \end{bmatrix} + z^2\begin{bmatrix} 0 \\ 2 \end{bmatrix} + z^1\begin{bmatrix} 1 \\ 1 \end{bmatrix} + z^0\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

The remaining partitions are Ω^6, Ω^7 and Ω^8 which is just the all "0" codeword.

$$\Omega^6 = \left[\begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 8 \end{bmatrix} \begin{bmatrix} 8 \\ 0 \end{bmatrix} \begin{bmatrix} 8 \\ 8 \end{bmatrix} \right]$$

and finally

$$\Omega^7 = \left[\begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 8 \\ 8 \end{bmatrix} \right], \Omega^8 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Fractional Dimensional Reed-Muller Composite Codes

This section describes how set partitioning is performed for trellis-coded fractional dimensional Reed-Muller composite codes. Instead of forming the L-fold Cartesian product of the same Reed-Muller codewords and $L_1 \times L_2 \times \ldots L_M$ Cartesian product is formed for a different constituent Reed-Muller codewords. Each constituent codeword can conceivably have a different lenght. Letting N_i denote the lenght of a constituent Reed-Muller code then the resulting codeword length after the Cartesian product is $N_m = \sum_{i=1}^{M} N_i$. For example let two constituent codewords be denoted as $C_1 = C(8,4,4)$, $C_2 = C(4,3,2)$. The $N_1 = 8, N_2 = 4$. The resulting code $C = C_1 \times C_2$ has codeword length $N_m = 12$. Under the previous construction, 2 (8,4,4) codes resulting in a 2-dimensional Reed-Muller code of length $N_m = 16$. The effective dimension of the L –fold Cartesian product is $L \times N$. Now the new code constructed composite Reed-Muller codes has a dimension that is smaller than the larger L –fold cartesian product form of the largest constituent code and larger than the L –fold cartesian of the smallest constituent code. This construction allows new codes to be constructed that have a richer set of code rates and weight spectra properties. For any Reed-Muller code denoted as $C_i = C(N_i, K_i, d_i)$. The overall rate R of the a new code is thus

$$R = \frac{\sum_{i=1}^{M} K_i - 1}{\sum_{i=1}^{M} N_i} \qquad \#$$

and the new code is denoted as

$$C_M = C\left(\sum_{i=1}^{M} N_i, \sum_{i=1}^{M} K_i - 1, d_{\text{free}} = * \right) \qquad \#$$

Continuing the current example, for $C_1 = C(8,4,4), C_2 = C(4,3,2)$ a new "fractional" dimension Reed-Muller is created which is $C_M = C(12,6,d_{\text{free}} = *)$. The resulting code has rate $R = \frac{1}{2}$, which is the same as the primary constitituent code with rate $R = \frac{1}{2}$. Furthermore the simple Reed-Muller constituent codes can be used for efficient encoding and decoding. Going a few steps further $C_1 = C(8,4,4), C_2 = C(4,3,2), C_3 = C(8,4,4)$ then $C_M = C(20, 10, d_{\text{free}} = *)$. Using the same codes but now $C_1 = C(8,4,4), C_2 = C(4,3,2), C_3 = C(4,3,2)$ yields $C_M = C(16,9,d_{\text{free}} = *)$. As shown a trelllis-coded, multidimensional Reed-Muller composite codes can be formed of new code rates. for systems that require a fixed or higher data user data rate, this construction will be used to meet this requirement. If it is desired to use rate 1/2 code for "heavy-coding" of a channel when the

constituent code is a $(8,4,4)$ Reed-Muller code then a new code can be constructed with $C_1 = C(8,4,4), C_2 = C(4,3,2)$ and $C_M = C(12,6,d_{\text{free}} = *)$. The resulting code has rate $R = \frac{1}{2}$. The properties of the new code are "inherited" from the parent constituent codes and the amount of "inheritance" can be adjusted by the set partitioning and the number of memory stages in the trellis-encoder. Since the code space of the smaller dimension constituent code is of smaller dimensionality than the largest constituent code, the set partitioning can be contrived to exploit the characteristics of each particular subspace. Table (ref: tab:rmcomp2) shows examples of trellis-coded multidimensional codes C_M that can be contructed from two composite Reed-Muller codes $C(8,4,4)$ and $C(4,3,2)$ for 2-fold and 3-fold cartesian products to form fractional dimensional composite codes. The minimum distance $(d_{\min})_M$ of the composite code is formed as $(d_{\min})_M = \sum_{i=1}^{M} (d_{\min})_i$ where $(d_{\min})_i$ is the minimum distance for each composite code. The resulting codes have code lengths that are not simple powers of 2 nor are the resulting code rates less than 1/2 which is a characteristic of most Reed-Muller codes.

[B]

C_1	C_2	C_3	C_M	$(d_{\min})_M$	Code Rate
$C(4,3,2)$	$C(4,3,2)$	–	$C(8,5,*)$	4	5/8
$C(4,3,2)$	$C(8,4,4)$	–	$C(12,6,*)$	6	1/2
$C(8,4,4)$	$C(8,4,4)$	–	$C(16,7,*)$	8	7/16
$C(4,3,2)$	$C(4,3,2)$	$C(4,3,2)$	$C(12,8,*)$	6	2/3
$C(4,3,2)$	$C(4,3,2)$	$C(8,4,4)$	$C(16,9,*)$	8	9/16
$C(4,3,2)$	$C(8,4,4)$	$C(8,4,4)$	$C(20,10,*)$	10	1/2
$C(8,4,4)$	$C(8,4,4)$	$C(8,4,4)$	$C(24,11,*)$	12	11/24

caption [E]

Table (ref: tab:rmcomp3) shows examples of trellis-coded multidimensional codes C_M that can be contructed from three composite Reed-Muller codes with $C(16,5,8)$ being the additional Reed-Muller code using 3-fold cartesian products.

[B]

C_1	C_2	C_3	C_M	$(d_{\min})_M$	Code Rate
$C(16,5,8)$	$C(4,3,2)$	$C(4,3,2)$	$C(24,10,*)$	12	5/8
$C(16,5,8)$	$C(4,3,2)$	$C(8,4,4)$	$C(28,11,*)$	14	11/28
$C(16,5,8)$	$C(8,4,4)$	$C(8,4,4)$	$C(32,12,*)$	16	3/8
$C(16,5,8)$	$C(16,5,8)$	$C(4,3,2)$	$C(36,12,*)$	18	1/3
$C(16,5,8)$	$C(16,5,8)$	$C(8,4,4)$	$C(40,13,*)$	20	13/40
$C(16,5,8)$	$C(16,5,8)$	$C(16,5,8)$	$C(48,14,*)$	24	7/24

caption [E]

In its simplest form, set partitoning for composite codes follows the same methodology as presented previously for new codes constructed with codes that are the same. To represent a composite code $C_M = C(8,4,4) \times C(4,3,2)$ Reed-Muller code, a 2×4 binary matrix is formed

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_1^3 & y_1^2 & y_1^1 & y_1^0 \\ - & y_2^2 & y_2^1 & y_2^0 \end{bmatrix} \quad \#$$

where y_1 represents a codeword label for $C(8,4,4)$ and y_2 represents a codeword label for $C(4,3,2)$. A total of $2^{3+4} = 127$ points (i.e. codeword constellation points) represent the unpartitioned signal set formed from the cartesian product. Each 2-tuple $[y_1, y_2]^T$ has two entries that represent a branch (i.e. Reed-Muller codeword) label pair in table (ref: tab:map). Each branch label can also be

represented in binary form in equation (ref: eq:2tuple) where $y_i^j \in \{0,1\}$ and $y_1 = 8y_1^3 + 4y_1^2 + 2y_1^1 + y_1^0$ and $y_2 = 4y_2^2 + 2y_2^1 + y_2^0$. The same mechanism for set partitioning cite: pietrobon2 is used with codes and subcodes of a $(2,2,1)$ linear block code. Therefore C_0 is a $(2,2,1)$ block code and C_1, C_2 are subcodes $(2,1,2)$ and $(2,0,\infty)$ respectively of C_0 and $C_0 \supset C_1 \supset C_2$. However an additional subcode is introduced $C_0^{'}$ which facilitates the fact that the space spanned by y_2 is of smaller dimension than the space spanned by y_1.

$C_0^{'} = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$ is a subcode of C_0 and is actually used to "supress" the fourth dimension of y_2 during set partitioning. Notice that $C_1 \not\subset C_0^{'}$ but $C_2 \subset C_0^{'}$ which implies that the benefit of partitioning

Level (p)	Partition Ω^p	MSSD (Δ_p^2)	Generator $(t_p)^T$
0	$\Omega(C_0^{'}, C_0, C_0, C_0)$	$\min(32,16,16,16) = 16$	$\begin{bmatrix} 0 & 1 \end{bmatrix}$
1	$\Omega(C_0^{'}, C_0, C_0, C_1)$	$\min(32,16,16,32) = 16$	$\begin{bmatrix} 1 & 1 \end{bmatrix}$
2	$\Omega(C_0^{'}, C_0, C_0, C_2)$	$\min(32,16,16,\infty) = 16$	$\begin{bmatrix} 0 & 2 \end{bmatrix}$
3	$\Omega(C_0^{'}, C_0, C_1, C_2)$	$\min(32,16,32,\infty) = 16$	$\begin{bmatrix} 2 & 2 \end{bmatrix}$
4	$\Omega(C_0^{'}, C_0, C_2, C_2)$	$\min(32,16,\infty,\infty) = 16$	$\begin{bmatrix} 0 & 4 \end{bmatrix}$
5	$\Omega(C_0^{'}, C_1, C_2, C_2)$	$\min(32,32,\infty,\infty) = 32$	$\begin{bmatrix} 4 & 4 \end{bmatrix}$
6	$\Omega(C_0^{'}, C_2, C_2, C_2)$	$\min(32,\infty,\infty,\infty) = 32$	$\begin{bmatrix} 8 & 0 \end{bmatrix}$
7	$\Omega(C_2, C_2, C_2, C_2)$	$\min(\infty,\infty,\infty,\infty) = \infty$	-

[B] caption [E]

Multidimensional Trellis Encoder/Decoder Design

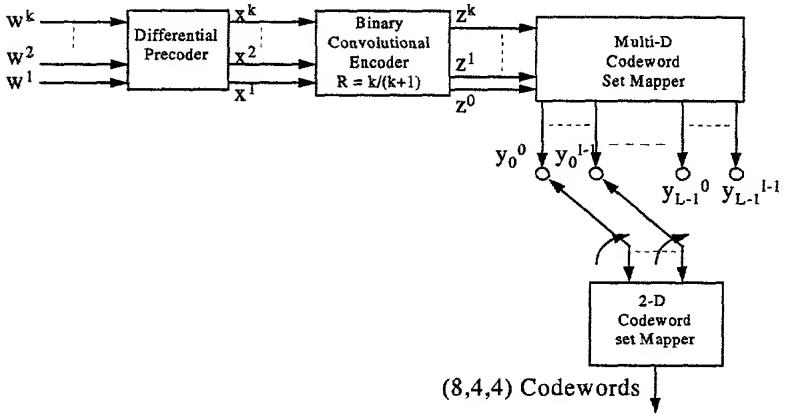
This section describes the design of the trellis encoder/decoder pair for both the multidimensional and coset design of trellis codes. Systematic methods based on cite: calderbank2 , cite: turgeon are used to design the convolutional encoder/decoder pairs in general. Issues such as rotational invariance are covered in cite: wei1 , cite: wei2 , cite: wei3 and cite: pietrobon3 . A trellis puncturing technique similar to cite: kim is necessary for the $(8,4,4)$ code if overall code rates of the new code is to be the same as the "1-D" Reed-Muller codes. A better technique based on composite Reed-Muller codes is the preferred approach for designing codes that provide for higher user data rates.

Encoder Designs

This section describes the design options and the design used to implement the encoder stage. Figure ref: fig:encoder shows the overall structure of the encoder as described in cite: pietrobon2 , cite: pietrobon1 . The key component parts of the encoder include (1) a differential precoder to preserve rotational invariance of signal sets (2) a rate $R = \frac{k}{k+1}$ convolutional encoder to provide redundancy in signal space (3) a multidimensional signal set mapper and (4) a

Reed-Muller signal set mapper. This design will also how each of these components are developed.

Encoder System



Encode System

There are some simplifications for design step (1) due to the characteristics of rotational invariance related to the underlying geometrical properties of Reed-Muller codes. Design steps (2) and (3) will be combined using the approach in cite: calderbank2 to devise the convolutional encoder by first using products-of-sums expansions to precisely define the convolutional encoder/signal mapper. "Optimal" branch assignments to encoder states will use the approach in cite: turgeon . Preliminary discussions of design step (4) were described in section ref: sect:Trellis .

The trellis encoder design will start with the set partitioning shown in table (ref: tab:trellis1Drm_2). This is because by defining the convolutional encoder with rate $R = \frac{k}{k+1}$, k represents the number of partition levels below partition level Ω^1. The parameter k should large enough to select a partition with a larger MSSD than Ω^1. Set partitioning yields a larger MSSD relative to Ω^1 at Ω^3 in table (ref: tab:trellis1Drm_2) while the MSSD does not increase until Ω^5 in table (ref: tab:trellis1Drm_2). Therefore $k \geq 2$ for the encoder. The next step is to determine the number of memory states which will be denoted as ν. Since the design rules will adhere to Ungerboeck's design rules cite: ungerboeck , then the number of states should be at least twice the number of states that the encoder can transition to at each time or $2^k \leq 2^{\nu-1}$ or $k \leq \nu - 1$, $\nu \geq 3$. There should be at least 8 encoder states. Figure ref: fig:trellisencoder shows a high level description of the trellis encoder.

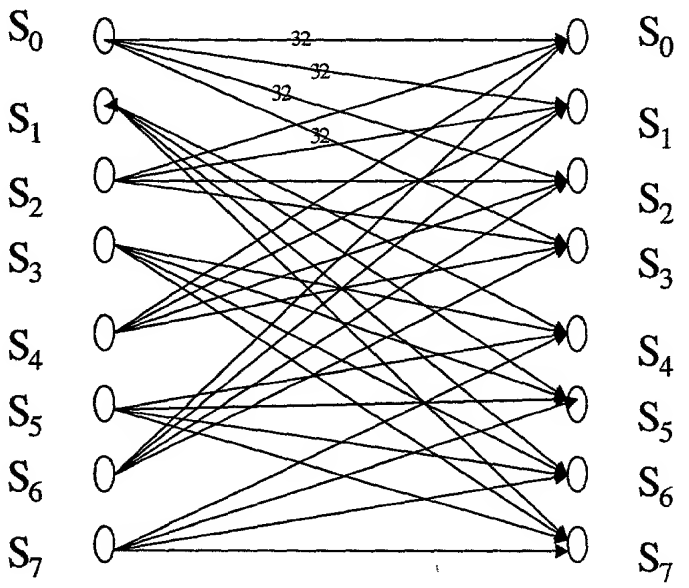
As a preview for the performance analysis, the maximum potential coding gain γ should be $\gamma = 3$ dB. because the new minimum distance for the trellis code is $2d_{\min} = 8$ of the trellis coded, 2-D (8,4,4) Reed-Muller code. For a rate $R = 1/2 = 8/16$ convolutional code with $\nu = 3$, the asymptotic coding gain for one of the "optimal" codes cite: lincostello is $\gamma = 1.76$ dB, $d_{\text{free}} = 5$. For $k = 6, \nu = 7$, the maximum potential coding gain turns out to be 6 dB. because the new minimum distance for the trellis code is $4d_{\min} = 16$ of the 2-D, (8,4,4) Reed-Muller code. For a comparable "optimal" convolutional code cite: lincostello with $\nu = 7$, $\gamma = 3.98$ dB, $d_{\text{free}} = 10$.

Each state represents a set of branch labels that are ordered to reflect how the label edges are

assigned for each state transition. Thus state $S_0 = \Omega_1^1$ and state $S_1 = \Omega_1^1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ where Ω_1^1 denotes the partition at level 1 from table (ref: tab:trellis1Drm_2). (Each partition will be denoted as Ω_1^p to distinguish from partitions defined for table (ref: tab:trellis1Drm)). The remaining states have the same tuples but the tuples are ordered differently in each state. Thus S_2, S_4, S_6 are composed of tuples in Ω_1^1 and S_1, S_3, S_5 are composed of tuples in $\Omega_1^1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. There are 128 branch label tuples in Ω_1^1 and since $k = 2$ there are 4 transitions from each state thus there are 32 parallel transitions (i.e. branch label tuples) from each state. (Only the transitions from state S_0 are shown for brevity.) Each transition from a state is an edge of the trellis and will be denoted as $B_{m,n}$ where m is the originating state and n is the branch transition for $n = 0, 1, 2, 3$. Thus $B_{0,0} = \Omega_1^3$ which is the partition for Ω_1^{1+k}. From table (ref: tab:trellis1Drm_2),

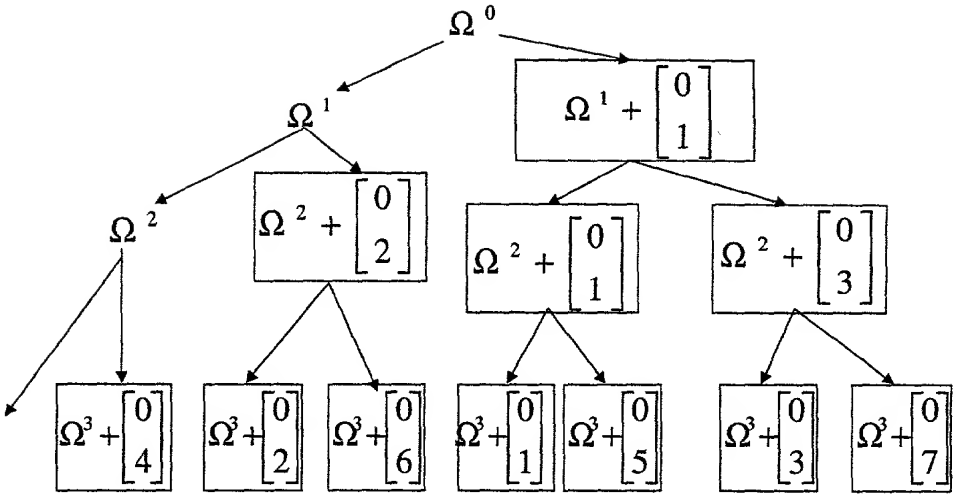
$$\Omega_1^3(z) = \Omega_1^3 + z^2 \begin{bmatrix} 0 \\ 4 \end{bmatrix} + z^1 \begin{bmatrix} 0 \\ 2 \end{bmatrix} + z^0 \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

2 x (8,4,4) Trellis encoder



Trellis Structure for $\nu = 3$ states and $k = 2$ inputs for encoder

Figure ref: fig:2dsetpart shows the set partitions for the 2-D trellis code. To get Ω_1^3 first Ω_1^1, Ω_1^2 must be formed. $\Omega^1 = \Omega(C_0, C_0, C_0, C_1)$ from table (ref: tab:trellis1Drm_2) is the same as Ω_1^1 from table (ref: tab:trellis1Drm). Equation (ref: eq:omega) shows that $\Omega^2 \neq \Omega_1^2$.



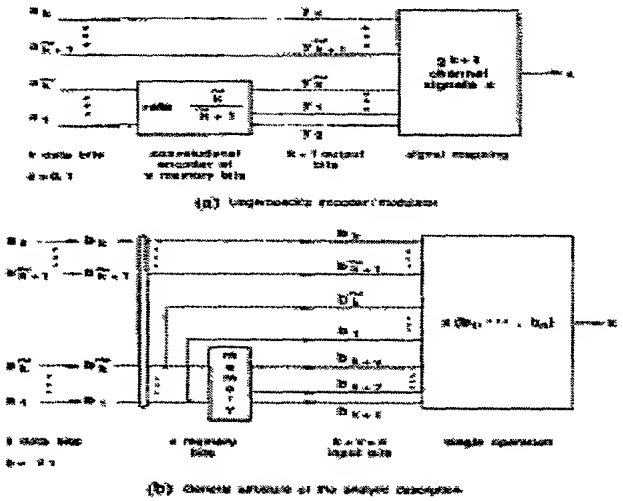
Set partitioning for $2 \times (8,4,4)$ Reed-Muller code, $i = 0,1,2,3$

The branch labels for S_0 then become $B_{0,0} = \Omega_1^3$, $B_{0,1} = \Omega_1^3(4)$, $B_{0,2} = \Omega_1^3(2)$, $B_{0,3} = \Omega_1^3(6)$. The labeling for each $B_{0,n}$ is an ordered set using additional rules described in cite: turgeon . Using these additional rules, the pairs of branch label tuples that are separated by maximum Euclidean distance are

Once the trellis states and transition labels are defined, the convolutional encoder can be designed. There are two established techniques to design the convolutional encoder. Figure ref: fig:ccdesign shows the architectures for the two approaches as summarized in cite: turgeon . The design of the encoding stage is driven by a set of parameters needed to compute BER performance cite: benedetto .

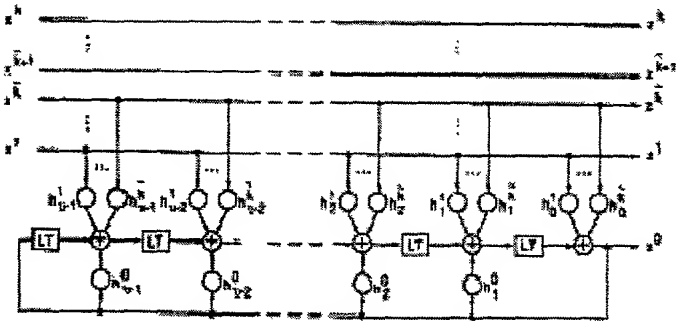
The convolutional encoders (see figure ref: fig:ccdesign) are usually implemented as systematic feedback or systematic, feedforward shift register realizations with modulo-2 addition. It has been shown that one can convert feedback realizations to feedforward and vice versa cite: porath2 . Figure ref: fig:ungccencode shows the architecture of the convolutional encoders used in cite: pietrobon1 . This is a systematic, feedback convolutional encoder with at most $v\left(\tilde{k}+1\right)$ encoder tap weights that must be determined from the trellis - which is already defined. There are $2^{v(\tilde{k}+1)}$ possible binary assignments to the tap weights and only $2^{\tilde{k}+v}$ state and edge label assignments. Therefore there are many systematic, feedback encoder realizations of the trellis. Note that there are some assignments that won't realize the desired trellis. Ungerboeck cite: ungerboeck devised schemes to perform the search for systematic, feedback encoder realizations which were modified slightly by Costello, et al cite: pietrobon2 . The measure is goodness is determined by equations (ref: eq:error1 , ref: eq:errorbit). Two very efficient algorithms were proposed cite: porath1 that exploit the growth of the column distance function (CDF) to allow a nested search instead of an exhaustive search. An even more efficient search algorithm is proposed by Costello, et al cite: malladi but using a systematic, *feedforward* encoder realization. However to get comparable d_{free} of systematic, feedback realizations requires constraint lengths that are $2v$ the size of the constraint length for feedback realizations. Another inherent drawback of these realizations is either feedforward or feedback is that these are linear codes which might not perform as well as nonlinear encoding schemes.

Trellis Encoder Designs



Convolutional Encoder Design Architectures

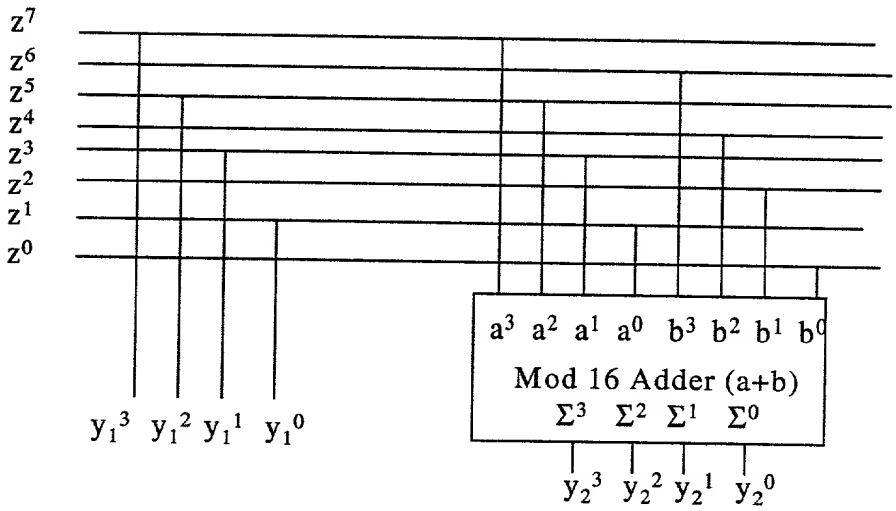
Ungerboeck Convolutional Encoder Designs



Sytematic Convolutional Encoder with feedback

Figure (ref: fig:sigmapper) shows the signal set mapper for coded bits $z = [z^7, z^6, \ldots, z^1, z^0]$ to y_1, y_2 where $y_1 = [y_1^3, y_1^2, y_1^1, y_1^0]$ and $y_2 = [y_2^3, y_2^2, y_2^1, y_2^0]$. The signal mapper represents the realization of equation (ref: eq:code2D). The two 4-bit outputs y_1, y_2 of the signal set mapper are then selected by the 2-D codeword selector to form two $(8,4,4)$ Reed-Muller codewords.

2 x (8,4,4) Signal Mapper



2 × (8,4,4) signal set mapper with modulo-16 addition.

The lower figure (see figure (ref: fig:ccdesign)) shows the approach devised in cite: calderbank1 , cite: calderbank2 . With this approach the signal mapping function $x(b_1, b_2, \ldots, b_{\tilde{k}}, b_{\tilde{k}+1}, \ldots, b_k, b_{k+1}, \ldots, b_{k+v})$ is a function of all inputs and outputs of memory elements with $b_i \in \{+1, -1\}$. (This approach also does not necessarily result in a linear "encoding" step as does the convolutional encoding approach.) The terms $b_1, b_2, \ldots, b_{\tilde{k}}$ are inputs that impact the behavior of the state machine realization of the convolutional code. These terms determine the number of transitions from each state where $2^{\tilde{k}}$ is the number of transitions. The terms $b_{k+1}, \ldots, b_{k+v}$ are the outputs of the memory elements. Letting $n = k + v$ the signal mapping function can be expanded in a sums of products form

$$x(b_1, b_2, \ldots, b_n) = d_0 + \sum_{i=1}^{n} d_i b_i + \sum_{i,j=1}^{n} d_{i,j} b_i b_j + \sum_{i,j,k=1}^{n} d_{i,j,k} b_i b_j b_k + \ldots + d_{1,2,\ldots,n} b_1 b_2 \ldots b_n \quad \#$$

Equation (ref: eq:sumprod) can be solved for $d_0, d_i, d_{i,j}, \ldots, d_{1,2,\ldots,n}$. Each product term can then be converted into a shift register and combinational logic realization of binary terms. The resulting configuration might not be as "optimal" in performance as some of Ungerboeck's approaches. When the output signal constellation point is an L-multidimensional, the coordinate label of a signal constellation point in each dimension has the form

$$x_k = x_k(b_1, b_2, \ldots, b_n), \; k = 1, 2, \ldots, L, \; b_i = \pm 1$$

Set element labeling is determined by using the "signal difference" rules established in cite: turgeon . As a quick review, given a 1 dimensional constellation (e.g. M-AM), a signal difference at a particular bit position of the mapping of input bits and encoder memory stages to output signal levels is defined as

$$\delta_i = |x(b_1, b_2, \ldots, b_{i-1}, b_i, \ldots, b_{k+v}) - x(b_1, b_2, \ldots, b_{i-1}, \overline{b_i}, \ldots, b_{k+v})|$$

For an L-multidimensional constellation, a signal difference vector for bit position b_i is $\boldsymbol{\delta}_i = (\delta_{i_1}, \delta_{i_2}, \delta_{i_3}, \ldots, \delta_{i_L})$ where

$$\delta_{i_m} = |x_m(b_1, b_2, \ldots, b_{i-1}, b_i, \ldots, b_{k+v}) - x_m(b_1, b_2, \ldots, b_{i-1}, \overline{b_i}, \ldots, b_{k+v})|$$

For $2 \times (8,4,4)$ Reed-Muller code then $\boldsymbol{\delta}_i = (\delta_{i_1}, \delta_{i_2})$. The definition of $\boldsymbol{\delta}_i$ is also modified from cite: turgeon since the constellation points in E_8 for Reed-Muller codewords don't have the same distance properties relative to constellation point labels like QAM or M-PSK. For example let $x_m(b_1, b_2, \ldots, b_{i-1}, \overline{b_i}, \ldots, b_{k+v}) = 0$, then for 16-QAM if $x_m(b_1, b_2, \ldots, b_{i-1}, b_i, \ldots, b_{k+v}) > x_m(b_1, b_2, \ldots, b_{i-1}, \overline{b_i}, \ldots, b_{k+v})$ then

$$\delta_{i_m} = \begin{cases} 4 \;, & |x_m(b_1, b_2, \ldots, b_{i-1}, b_i, \ldots, b_{k+v}) - x_m(b_1, b_2, \ldots, b_{i-1}, \overline{b_i}, \ldots, b_{k+v})| \bmod(16) < 8 \\ 8 \;, & |x_m(b_1, b_2, \ldots, b_{i-1}, b_i, \ldots, b_{k+v}) - x_m(b_1, b_2, \ldots, b_{i-1}, \overline{b_i}, \ldots, b_{k+v})| \bmod(16) \geq 8 \end{cases} \quad \#$$

Equation (ref: eq:sigdiffrm) reflects that many signal labels map into the same distance thus the same signal differences. This redefinition should simply the ordering requirements on signal labels that are assigned to trellis states. Figure (ref: fig:encoderI) shows an encoder realization for $\widetilde{k} = 2$ inputs (i.e. b_1, b_2) that cause the transitions from each state of the trellis in figure (ref: fig:encoder). There are 3 encoder delay elements with outputs denoted as b_8, b_9, b_{10}. The number of parallel transitions per edge transition is determined by 5 bits $b_2, b_4, \ldots, b_7$ or 32 parallel transitions. The branch labels are the outputs of the mapper function $x_i = x_i(b_1, b_2, \ldots, b_7, b_8, b_9, b_{10})$, $i = 1, 2$. Let each state σ be defined as $\sigma_{(a_{10}, a_9, a_8)_8} = \sigma(4a_{10} + 2a_9 + a_8)$ where

$$a_i = \frac{1 - b_i}{2}$$

The branch edge labels for even number states $\sigma_{0,2,4,6}$ are formed as follows. Let $B_{0,2a_1+a_2} = \Omega_1^3(2a_2 + a_1)$ where "0" is the state number and a_1, a_2 are the inputs bits that cause state transitions. Therefore in vector form

$$B_{0,*} = \begin{bmatrix} \Omega_1^3(0) \\ \Omega_1^3(4) \\ \Omega_1^3(2) \\ \Omega_1^3(6) \end{bmatrix} \quad \#$$

where $B_{0,*}$ is a vector of all transitions. The branch labels for state "2" are formed as

$$B_{2,*} = \Pi \otimes \begin{bmatrix} \Omega_1^3(0) \\ \Omega_1^3(4) \\ \Omega_1^3(2) \\ \Omega_1^3(6) \end{bmatrix} = \begin{bmatrix} \Omega_1^3(4) \\ \Omega_1^3(2) \\ \Omega_1^3(6) \\ \Omega_1^3(0) \end{bmatrix} \quad \#$$

where Π is a permutation matrix of the form

$$\Pi = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and $\otimes$ is the Kronecker product since Ω are sets and the goal is to permute the sets of elements. Likewise

$$B_{4,*} = \Pi^2 \otimes B_{0,*}, B_{6,*} = \Pi^3 \otimes B_{0,*} \quad \#$$

for the odd-numbered states the construction is the same which yields

$$B_{1,*} = \begin{bmatrix} \Omega_1^3(1) \\ \Omega_1^3(5) \\ \Omega_1^3(3) \\ \Omega_1^3(7) \end{bmatrix} \quad \#$$

where $B_{1,2a_1+a_2} = \Omega_1^3(2a_2 + a_1) + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and Π^i is normal matrix multiplication.

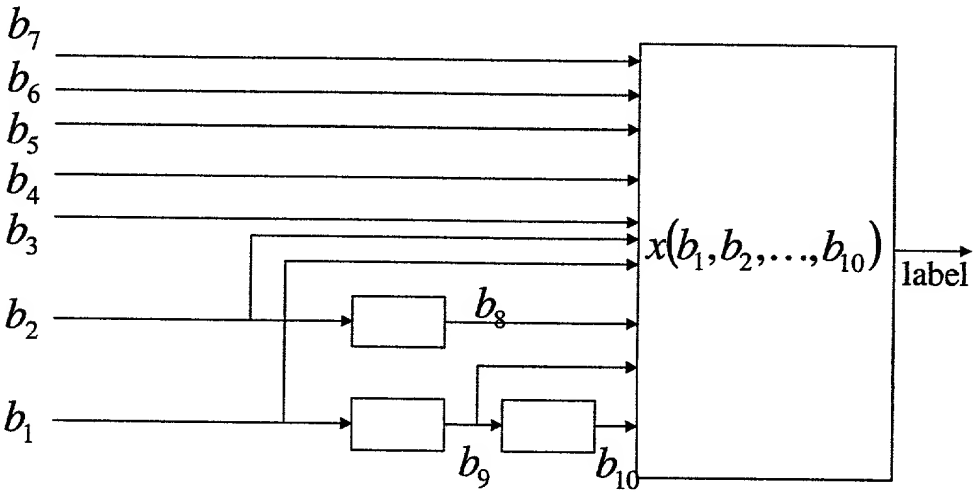
$$B_{3,*} = \Pi^1 \otimes B_{1,*}, B_{5,*} = \Pi^2 \otimes B_{1,*}, B_{7,*} = \Pi^3 \otimes B_{1,*}, \quad \#$$

Now that each state is defined by tuple $\sigma(b_{10}, b_9, b_8)$ and each branch transition is defined by equations (ref: eq:br1 - ref: eq:br5), equation (ref: eq:sumprod) can be solved for constants $d_0, d_i, d_{ij}, \ldots, d_{1,2,\ldots n}$ for each dimensional - which is two.

Given that all trellis branches and state transitions to and from all states are defined a FSM can be constructed. The FSM was realized as a large mapping table of the form shown in table (ref: tab:FSMtrellis). For $\widetilde{k} = 2$, inputs bits b_1, b_2 trigger transitions from a current state to a new state. Before going to a new state along the appropriate transition, one tuple is selected from the coset - at the current state - representing all possible edge labels using $k - \widetilde{k}$ bits b_3, b_4, b_5.

Current State	$\tilde{k}$	Next State	Edge Transition Labels
000	00	000	$\Omega^3(0)$
000	01	001	$\Omega^3(4)$
000	10	010	$\Omega^3(2)$
000	11	011	$\Omega^3(6)$
001	00	100	$\Omega^3(1)$
001	01	101	$\Omega^3(5)$
001	10	110	$\Omega^3(3)$
001	11	111	$\Omega^3(7)$
010	00	000	$\Omega^3(4)$
010	01	001	$\Omega^3(2)$
010	10	010	$\Omega^3(6)$
010	11	011	$\Omega^3(0)$
⋮	⋮	⋮	⋮
111	00	100	$\Omega^3(7)$
111	01	101	$\Omega^3(1)$
111	10	110	$\Omega^3(5)$
111	11	111	$\Omega^3(3)$

[B] caption [E]



Encoder Realization I ($k = 2, v = 3$)

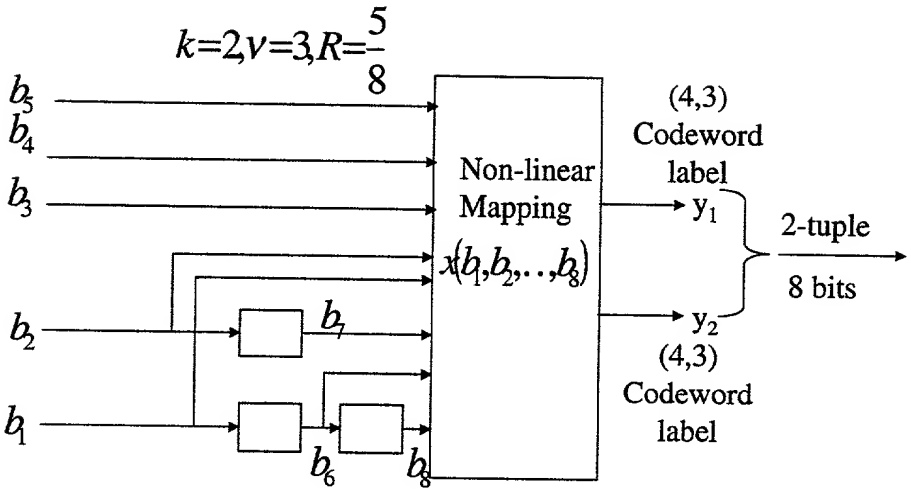
The design options have been discussed and now the particular implementation approach is described. Although the Ungerboeck approach used convolutional encoders to develop both linear and non-linear convolutional codes, the current implementation uses the non-linear approach. As for encoding there are two techniques. One technique is to solve for the parameters d and then for each input codeword to compute a branch label tuple by multiplying the corresponding row of the Hadamard matrix H for all possible bit combinations by d. The encoding steps are

1. At time t, form row vector in equation (ref: eq:encodehad) of product terms with each set of input message bits $[b_1, b_2, \ldots, b_k]$ and memory states $[b_{k+1}, b_{k+2}, \ldots, b_{k+v}]$ of the trellis FSM. The sums of products expansion is shown in equation (ref: eq:sumprod).
2. Perform vector-matrix multiplication in equation (ref: eq:encodehad) to form branch label 2-tuple $\begin{bmatrix} x_1 & x_2 \end{bmatrix}$
3. Convert branch label 2-tuple to Reed-Muller binary codewords as per table (ref: tab:map).
4. Modulate binary codewords to BPSK or QPSK constellation points
5. Use FSM for trellis to compute next state at time $t+1$ or $(b_{k+1}, b_{k+2}, \ldots, b_{k+v})_{t+1} = \sigma\{(b_{k+1}, b_{k+2}, \ldots, b_{k+v})_t; b_1, b_2, \ldots, b_k\}$
6. Go to step (1).

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} 1 & b_1 & \cdots & b_1 b_2 \ldots b_n \end{bmatrix} \begin{bmatrix} d_{0,1} & d_{0,2} \\ d_{1,1} & d_{1,2} \\ \vdots & \vdots \\ d_{(1,2,\ldots n),1} & d_{(1,2,\ldots n),2} \end{bmatrix} \quad \#$$

Decoder Design

This section describes the decoder design. Figure (ref: fig:encoder_4_3) and (ref: fig:decoder_4_3) show the encoder/decoder pair for the case under investigation. The encoder has $k = 5$ input bits and 8 output bits. The 8 output bits are BPSK or QPSK modulated for each codeword. This particular configuration has $\tilde{k} = 2$ input bits out of $k = 5$ bits used for a trellis finite state machine (FSM) designed with $v = 3$ memory cells. The non-linear mapping function uses 8 input bits $b_1, b_2, \ldots, b_5, b_6, b_7, b_8$ to generate a 2-tuple branch label for the next state from a current state defined by b_6, b_7, b_8. The number of transitions from a current state is $2^{\tilde{k}} = 4$. The number of branch tuples per edge transition is determined by bits b_3, b_4 and b_5. There are $2^{k-\tilde{k}} = 8$ possible parallel transition per edge in the trellis. For each branch label in each 2-tuple, a (4,3) Reed-Muller codeword is formed for each branch label which is then translated into a BPSK or QPSK signal. The overall code rate is $R = \frac{5}{8}$.

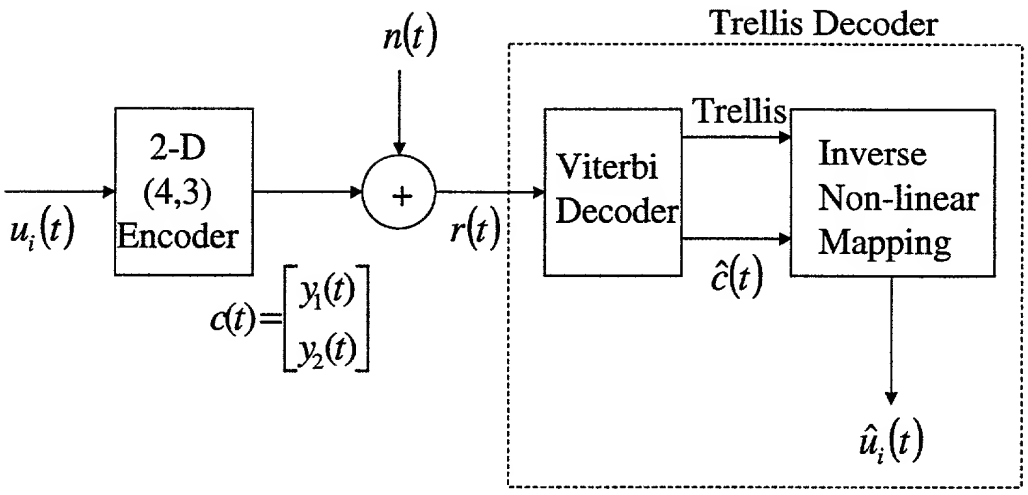


Encoder for 2-D (4,3) Reed-Muller code

Figure (ref: fig:decoder_4_3) shows the decoder for the 2-D (4,3) Reed-Muller code. Input message bits with $u_i(t) = [b_1(t), b_2(t), \ldots, b_5(t)]$ are encoded into codeword 2-tuples $c(t) = [y_1(t), y_2(t)]^T$ where each $y_1(t), y_2(t)$ is a (4,3) Reed-Muller codeword. These codewords are modulated and transmitted over an AWGN channel with noise $n(t)$. The Viterbi decoder receives channel signal $r(t)$ and forms estimates of channel codewords $\hat{c}(t)$ and stores accompanying state progression information in a trellis. The inverse non-linear mapping function uses both the trellis and the decoded codewords $\hat{c}(t)$ to estimate the input message sequence $\hat{u}_i(t)$. The trellis contains a sequence of states and a set of transitions and codeword vectors where each state contains information such as

- The transitions between states (i.e. bits $\hat{b}_1(t), \hat{b}_2(t)$) at each time epoch for the codeword sequence $\hat{c}(t)$.
- The current and next states for codeword sequence $\hat{c}(t)$ (i.e. bits $\hat{b}_6(t), \hat{b}_7(t), \hat{b}_8(t)$)
- The address label for the 2-tuple selected for estimated codeword $\hat{c}(t)$, (i.e. bits $\hat{b}_3(t), \hat{b}_4(t), \hat{b}_5(t)$)

With this information, the estimate of input message by the non-linear mapping becomes $\hat{u}_i(t) = \left[\hat{b}_1(t), \hat{b}_2(t), \ldots, \hat{b}_5(t)\right]$ at each time epoch in the trellis.



Decoder for 2-D (4,3) Reed-Muller code

References

wicker S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice-Hall, Inc., 1995.

mceliece R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1996.

lincostello S. Lin and D. J. Costello, *Error Control Coding : Fundamentals and Applications*, Prentice-Hall, Inc., 1983.

wolf J. K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis," *IEEE Transactions on Information Theory*, vol. IT-24, No. 1, January 1978.

wei4 L. Wei, "Trellis-coded Modulation with Multidimensional Constellations," *IEEE Transactions on Information Theory*, vol. IT-33, No. 4, July 1987.

pietrobon1 S. S. Pietrobon and D. J. Costello, Jr., "Trellis Coding with Multidimensional QAM Signal Sets", *IEEE Transactions on Information Theory*, vol. 39, No. 2, March 1993.

pietrobon2 S. S. Pietrobon, R. H. Deng, A. Lafanechere, G. Ungerboeck and D. J. Costello, Jr., "Trellis-Coded Multidimensional Phase Modulation", *IEEE Transactions on Information Theory*, vol. 36, No. 1, January 1990.

forney2 G. D. Forney, "Coset codes - Part I : Introduction and Geometrical Classification", *IEEE Transactions on Information Theory*, vol. 34, 1988, No 5, pp. 1123-1151.

forney3 G. D. Forney, "Coset codes - Part II : Binary Lattices and Related Codes", *IEEE Transactions on Information Theory*, vol. 34, No 5, 1988, pp. 1152-1187.

calderbank1 A. R. Calderbank and N. J. A. Sloane, "New Trellis Codes Based on Lattices and Cosets," *IEEE Transactions on Information Theory*, vol. IT-33, No 2, March 1987.

calderbank2 A. R. Calderbank and J. E. Mazo, "A New Description of Trellis Codes," *IEEE Transactions on Information Theory*, vol. IT-30, No 6, November 1984.

turgeon J. M. Turgeon and P. J. McLane, "Minimal Transmitter Complexity Design of Analytically Described Trellis Codes," *IEEE Transactions on Communications*, vol. 38, No. 9, September 1990.

ungerboeck G. Ungerboeck, "Channel Coding with Multi/Phase Signals," Evaluation of the Performance of Error-Correcting Codes on a Gilbert Channel," *IEEE Transactions on Information Theory*, vol. IT-28, No 1, January 1982.

benedetto S. Benedetto, M. Mondin and G. Montorsi, "Performance Evaluation of Trellis-coded Modulation Schemes," *Proceedings of the IEEE*, vol. 82, No. 6, June 1994.

malladi S. S. Malladi, F. Wang, D. J. Costello, Jr. and H. Ferreira, "Construction of Trellis Codes with a Good Distance Profile," *IEEE Transactions on Communications*, vol. 42, No. 2/3/4, Feb/March/April 1994.

porath1 J. Porath and T. Aulin, "Algorithm Construction of Trellis Codes," *IEEE Transactions on Communications*, vol. 41, No. 5, May 1993.

porath2 J. Porath, "Algorithms for Converting Convolutional Codes from Feedback to Feedforward Form and Vice Versa," *Electronic Letters*, vol. 25, No. 15, July 20, 1989.

chaib J. Chaib and H. Lieb, "Very Low Rate Trellis/Reed-Muller (TRM) Codes," *IEEE Transactions on Communications*, vol. 47, No. 10, October 1999.

sloane1 A. R. Calderbank and N. J. A. Sloane, "Four Dimensional Modulation with Eight-State Trellis Code," *AT&T Technical Journal*, vol. 64, No. 5, May-June 1985.

sloane2 A. R. Calderbank and N. J. A. Sloane, "An Eight-Dimensional Trellis Code," *Proceeding of the IEEE*, vol. 74, No. 5, May 1986.

lang G. R. Lang and F. M. Longstaff, "A Leech Lattice Modem," *IEEE Journal on Selected Areas in Communications*, vol. 7, No. 6, August 1989.

beery Y. Be'ery, B. Shahar and J. Snyders, "Fast Decoding of the Leech Lattice," *IEEE Journal on Selected Areas in Communications*, vol. 7, No. 6, August 1989.

kim J. Kim and G. J. Pottie, "On Punctured Trellis-coded Modulation," *IEEE Transactions on Information Theory*, vol. 42, No 2, March 1996.

wei1 L. Wei, "Rotationally Invariant Convolutional Channel Coding with Expanded Signal Space - Part I : 180^0," *IEEE Journal on Selected Areas in Communications*, vol. SAC-2, No. 5, September 1984.

wei2 L. Wei, "Rotationally Invariant Convolutional Channel Coding with Expanded Signal Space - Part II : Nonlinear Codes," *IEEE Journal on Selected Areas in Communications*, vol. SAC-2, No. 5, September 1984.

pietrobon3 S. S. Pietrobon, G. Ungerboeck, L. C. Perez and D. J. Costello, Jr., "Rotationally Invariant Nonlinear Trellis codes for Two-Dimensional Modulation," *IEEE Transactions on Information Theory*, vol. 40, No. 6, November 1994.

wei3 L. Wei, "Rotationally Invariant Trellis-Coded Modulations with Multidimensional *M*-PSK," *IEEE Journal on Selected Areas in Communications*, vol. 7, No. 9, December 1989.

bhargava V. K. Bhargava, D. Haccoun, R. Matyas and P. P. Nuspl, *Digital Communications by Satellite : Modulation, Multiple Access and Coding*, Krieger Publishing Company; Malabar, Florida, 1991.